UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/480,309	01/10/2000	DAVID N. WILNER	11283/2	4170

7590 08/14/2003

KENYON & KENYON
333 W SAN CARLOS STREET
SUITE 600
SAN JOSE, CA 95110-2711

EXAMINER	
ALI, SYED J	

ART UNIT	PAPER NUMBER
2127	

DATE MAILED: 08/14/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/480,309	WILNER ET AL.	
	Examiner	Art Unit	
	Syed J Ali	2127	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *June 6, 2003* .

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-38* is/are pending in the application.

4a) Of the above claim(s) *35-38* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-34* is/are rejected.

7)☒ Claim(s) *1,10 and 14* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

11)☐ The proposed drawing correction filed on _____ is: a)☐ approved b)☐ disapproved by the Examiner.

If approved, corrected drawings are required in reply to this Office action.

12)☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

13)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____ .

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).

a) ☐ The translation of the foreign language provisional application has been received.

15)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☒ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO-1449) Paper No(s) *4,5,8* .
4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: .

DETAILED ACTION

Election/Restrictions

1. Applicant's election without traverse of Group I (claims 1-34) in Paper No. 10 is acknowledged.

The requirement is still deemed proper and is therefore made FINAL.

2. Claims 35-38 are withdrawn from further consideration pursuant to 37 CFR 1.142(b) as being drawn to a nonelected invention, there being no allowable generic or linking claim. Election was made **without** traverse in Paper No. 10.

Claim Objections

3. Claims 1 and 14 are objected to because of the following informalities: Claim 1 recites the limitation "the reference symbol" in line 4. It is interpreted that this limitation is meant to refer to "the symbol reference", which would make the limitation consistent with the remainder of the claims. Furthermore, claim 14 recites the limitation, "referring to an a subsequent instruction...", which is grammatically incorrect. It appears that a deletion of the word "an" would correct this deficiency. Appropriate correction is required.

4. Claim 10 is objected to under 37 CFR 1.75(c), as being of improper dependent form for failing to further limit the subject matter of a previous claim. Applicant is required to cancel the claim(s), or amend the claim(s) to place the claim(s) in proper dependent form, or rewrite the claim(s) in independent form.

Specifically, the limitation of claim 10 recites, "the link stub includes a jump instruction to the external location." This is considered to be analogous to the limitation of claim 8 that recites, "executing a jump instruction in the number of instructions that refers to a link stub corresponding to an external location in a second domain." If the jump instruction of claim 10 refers to a different jump instruction than the jump instruction of claim 8, such a distinction is unclear as the claims are presented, and a different identifier is required to make the distinction clearer.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

6. Claims 1-3, 8-10, 15-19, and 21-23 are rejected under 35 U.S.C. 102(b) as being anticipated by Kempf et al. (USPN 5,359,721) (hereinafter Kempf).

As per claim 1, Kempf discloses a method, comprising the steps of:

loading a code module into a memory space of a first domain (col. 6 line 60 - col. 7 line 22, "During a mapping operation, the program code and/or the group of data is inserted into the

address space in bulk, and the addresses of the address space is associated with the program code and/or data"), the code module including an instruction having a symbol reference (col. 6 line 60 - col. 7 line 22, "Additionally, during a mapping operation, symbol address information about a program segment is obtained from the program segment by accessing the locations where the program segment is deposited");

determining if the reference symbol is to an external location outside of the memory space (col. 9 lines 3-13, "If the code table indicates that the program code segment has not been linked in the address space, the client then obtains the program code segment object from a program code manager");

generating a link stub for the symbol reference when the symbol reference is to an external location to access the external location (col. 9 lines 14-25, "The client process then maps the program code segment into the address space and into its own address space and links the program code segment relative to addresses in the executing process"); and

redirecting the instruction to the link stub (col. 9 lines 14-25, "Upon linking the program code segment to the executing process, the client process then locates the initialization function of the program code segment, block 100. Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 2, Kempf discloses the method of claim 1, wherein the link stub is part of a linking table entry corresponding to the symbol reference (col. 7 lines 23-44, "The code table comprises entries for each linked program code segment for the address space").

As per claim 3, Kempf discloses the method of claim 1, wherein the link stub is a jump instruction to the external location (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 8, Kempf discloses a method, comprising:

creating a task in a first domain, the task executing a number of instructions (col. 6 line 60 - col. 7 line 22, "During a mapping operation, the program code and/or the group of data is inserted into the address space in bulk, and the addresses of the address space is associated with the program code and/or data");

executing a jump instruction in the number of instructions that refers to a link stub corresponding to an external location in a second domain (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function");

executing the link stub (col. 9 lines 14-25, "Upon linking the program code segment to the executing process, the client process then locates the initialization function of the program code segment, block 100. Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 9, Kempf discloses the method of claim 8, wherein the link stub is part of linking table entry corresponding to the external location (col. 7 lines 23-44, "The code table comprises entries for each linked program code segment for the address space").

As per claim 10, Kempf discloses the method of claim 8, wherein the link stub includes a jump instruction to the external location (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 15, Kempf discloses a computer system, comprising:

a system space having a number of memory locations (col. 6 line 60 - col. 7 line 9, "The address space object 62 comprises an address space (not shown) and provides an object oriented interface for performing limited operations on the address space on behalf of client processes executing in non-supervisor mode. The limited operations comprise an operation for returning a list of memory objects mapped into the address space and the corresponding base addresses for the memory objects");

a number of protection domains, at least one of the number of protection domains owning a portion of the system space (col. 6 line 60 - col. 7 line 9, "The limited operations comprise...an operation of mapping a program code segment and/or a group of data into the address space, producing a memory object corresponding to the mapped memory").

As per claim 16, Kempf discloses the computer system of claim 15, wherein the at least one of the number of protection domains includes at least one of:

a code module (col. 6 line 60 - col. 7 line 9, "During a mapping operation, the program code and/or the group of data is inserted into the address space in bulk, and the addresses of the address space is associated with the program code and/or data");

a link stub (col. 9 lines 14-25, "The client process then maps the program code segment into the address space and into its own address space and links the program code segment relative to addresses in the executing process"); and

an entry point (col. 9 lines 14-25, "Upon linking the program code segment to the executing process, the client process then locates the initialization function of the program code segment, block 100. Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 17, Kempf discloses the system of claim 16, wherein the link stub is part of a linking table entry in a linking table (col. 7 lines 23-44, "The code table comprises entries for each linked program code segment for the address space").

As per claim 18, Kempf discloses the system of claim 16, wherein the entry point is represented in a symbol table (col. 9 lines 14-25, "Upon linking the program code segment to the executing process, the client process then locates the initialization function of the program code segment, block 100").

As per claim 19, Kempf discloses 19 the system of claim 16, wherein at least one of the number of protection domains is a system protection domain that includes:

at least one code module including executable code for operating system services (col. 5 lines 36-54, "The system software 30 comprises an operating system"); and

at least one system object owned by the system protection domain (col. 5 lines 36-54, "Operating system 32 provides...an object oriented interface for securely mapping files into address spaces on behalf of a process executing in non-supervisor mode").

As per claim 21, Kempf discloses the system of claim 19, wherein at least one of the number of protection domains is a first protection domain that includes:

at least one code module including executable code for a first set of functions (col. 6 line 60 - col. 7 line 9, "During a mapping operation, the program code and/or the group of data is inserted into the address space in bulk, and the addresses of the address space is associated with the program code and/or data"); and

a number of link stubs (col. 9 lines 14-25, "The client process then maps the program code segment into the address space and into its own address space and links the program code segment relative to addresses in the executing process");

wherein at least one of the link stubs corresponds to a symbol referenced in the executable code for the first set of functions (col. 6 line 60 - col. 7 line 22, "Additionally, during a mapping operation, symbol address information about a program segment is obtained from the program segment by accessing the locations where the program segment is deposited"), and such

at least one link stub includes executable code to direct execution to the executable code for operating system services (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 22, Kempf discloses the system of claim 21, wherein the number of protection domains includes a second protection domain that includes:

at least one code module including executable code for a second set of functions (col. 9 lines 14-25, "The client process then maps the program code segment into the address space and into its own address space and links the program code segment", wherein the program code segment being linked is in another namespace, or protection domain); and

a number of entry points (col. 9 lines 14-25, "the client process then locates the initialization function of the program code segment", wherein the initialization function is the entry point);

wherein each of the entry points corresponds to a symbol in the executable code for the second set of functions (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 23, Kempf discloses the system of claim 22, wherein one of the link stubs of the first protection domain corresponds to one of the number of entry points in the second protection domain, and such link stub includes executable code to direct execution to the one of

the number of entry points in the second protection domain (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

7. Claims 28-29 and 32-34 are rejected under 35 U.S.C. 102(e) as being anticipated by Van Doorn (USPN 6,546,546).

As per claim 28, Van Doorn discloses a protection domain, comprising:

a memory space (col. 4 lines 55-62, "Memory management is separated into physical and virtual memory management", wherein the memory space is mapped to different protection domains and all management of this memory space is done by the Java Virtual Machine); and

a protection view designating a set of protection domains for unrestricted memory access (col. 5 lines 17-26, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains").

As per claim 29, Van Doorn discloses the protection domain of claim 28, further comprising:

a number of code modules, each of the number of code modules including at least one of executable code and data structures (col. 3 lines 48-65, "The present invention is most advantageously applied to the integration of a programming language system and an operating

system when the programming language exhibits the following system properties: 1. a well defined unit of protection embedded in the language. These units form the basis of the protection. Examples of these units are classes, objects, and modules").

As per claim 32, Van Doorn discloses the protection domain of claim 28, further comprising:

authorization information for enabling attachment of other protection domains (col. 8 line 63 - col. 9 line 6, "the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains").

As per claim 33, Van Doorn discloses the protection domain of claim 28, further comprising:

a task, having a task control block (this is handled by the Java runtime context), a task protection view (col. 5 lines 17-27, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains"), a protection switch stack (col. 4 lines 36-54, "A handler consists of a protection domain identifier, the address of a call-back function, and a stack pointer", wherein the stack pointer identifies all the namespaces within the protection view of the protection domain, and is located through the use of the protection domain identifier) and a task privilege level (col. 8 line 63 - col. 9 line 6, "the memory available to all protection domains is

mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains").

As per claim 34, Van Doorn discloses the protection domain of claim 28, further comprising:

protection domain attributes, including an indication whether the protection domain may be attached (col. 8 line 63 - col. 9 line 6, "the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains").

Claim Rejections - 35 USC § 103

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

9. Claims 4-7, 11, 20, 24-27, and 30-31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kempf in view of Van Doorn.

As per claim 4, Van Doorn discloses the following limitations not shown by Kempf, specifically the method of claim 1, further comprising the steps of:

determining if the external location is within a second domain that is within a protection view of the first domain (col. 5 lines 17-27, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains");

requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain (col. 8 line 63 - col. 9 line 6, "the memory available to all protection domains is mapped into the Java Nucleus with read/write permissions. This allows it to quickly access the data in different protection domains", wherein when a first domain requires access to methods of a second domain, the Java Nucleus performs the mapping, or attachment, process);

attaching the second domain to the first domain using an attachment mechanism (col. 9 line 7-23, "memory Region 0 is mapped into the executable content context 330. Part of this memory contains executable code and has the execute privilege associated with it").

It would have been obvious to one of ordinary skill in the art to add Van Doorn to Kempf since under certain circumstances, a code module may need to access data that is outside of its particular protection domain. In such cases, security measures typically do not allow such access since it may cause a breach. This is a deficiency of Kempf. Van Doorn makes up for this deficiency by allowing a protection domain to add other existing domains to its data protection set, thereby increasing the range of the code, while maintaining security measures.

As per claim 5, Van Doorn discloses the method of claim 4, further comprising the steps of:

determining whether the attachment request is permitted based on authorization information provided by the first domain (col. 8 line 63 - col. 9 line 6, "the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains");

wherein attachment to the second domain is not permitted when the attachment request is not permitted (col. 8 line 63 - col. 9 line 6, "the memory available to all protection domains is mapped into the Java Nucleus with read/write permission. This allows it to quickly access the data in different protection domains", wherein if the memory is not mapped to the Java Nucleus, attachment is not allowed since it may cause a security breach or a page fault).

As per claim 6, Van Doorn discloses the method of claim 4, wherein the attachment mechanism comprises adding the second domain to the protection view of the first domain (col. 9 line 7-23, "memory Region 0 is mapped into the executable content context 330. Part of this memory contains executable code and has the execute privilege associated with it") and Kempf discloses including a jump instruction to the external location in the link stub (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

As per claim 7, Kempf discloses 7 the method of claim 4, wherein the attachment mechanism comprises including a jump instruction to the external location in the link stub without altering the protection view of the first domain (col. 9 lines 14-25, "Upon locating the

initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function", wherein nothing is altered in the protection view of the first domain; rather, control is transferred to the location in memory of the desired instruction).

As per claim 11, Van Doorn discloses the following limitations not shown by Kempf, specifically, the method of claim 8, further comprising the steps of:

comparing the external location to a task protection view (col. 9 lines 41-60, "Consider the protection Domains A and B and a Method M which resides in Domain-B", wherein a determination is made as to whether or not the method called is within either the protection domain or the protection view from the domain that it is called from); and

generating a processing exception when the external location is outside the task protection view (col. 9 lines 41-60, "An instruction fault occurs when A calls Method M, since M is not mapped into context A").

It would have been obvious to one of ordinary skill in the art to add Van Doorn to Kempf since under circumstances in which a method outside of the protection view of the first domain is called, security may be breached by allowing methods to run that are not secured. Either securing the second domain, or denying access to the second domain can handle this. Van Doorn discloses denying access to the second domain when it is not mapped into the context of the first domain, thereby allowing a secure execution environment.

As per claim 20, Van Doorn discloses the system of claim 19, wherein the system protection domain includes a protection domain list that includes entries for each of the number of protection domains (col. 5 lines 17-27, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains").

As per claim 24, Van Doorn discloses the system of claim 16, wherein each of the number of protection domains includes a protection view defining a set of the number of protection domains to which unprotected access may be made (col. 5 lines 17-27, "Each protection domain has a view of its own subtree of the name space, the kernel address space has a view of the entire tree including all the subtrees of different protection domains").

As per claim 25, Van Doorn discloses the system of claim 24, wherein the protection view sets a memory range of allowable memory accesses, and wherein a memory fault is generated when memory access is attempted outside of the memory range (col. 5 lines 1-10, "The protection domain's virtual memory space is managed by the virtual memory service", "Each virtual page has associated with it a fault event that is raised, if set, when a fault occurs on an address within that page").

As per claim 26, Van Doorn discloses the system of claim 25, wherein the memory range is contiguous (Fig. 3, elements 310, 320, and 330, wherein the virtual memory pages being mapped are contiguous portions of memory).

As per claim 27, Van Doorn discloses the system of claim 25, further comprising an exception handling routine that is executed on the occurrence of the memory fault, the exception handling routine including a protection switch mechanism (col. 5 lines 11-16, "The event handlers for a virtual page fault do not have to reside in the same protection domain where the fault occurs", wherein the exception handler executes by finding the requested memory addresses from memory, thereby switching the protection view from only looking at the present protection domain to all memory domains that the Java Nucleus has access to)

As per claim 30, Kempf shows the following limitations not shown by Van Doorn, specifically the protection domain of claim 29, further comprising:

a symbol table comprising a number of symbol entries, at least one of the symbol entries specifying an entry point address within the memory space (col. 3 lines 14-22, "the address space manager is also used to provide a code table for identifying the first program code segment as not being linked in the second process, and to provide symbol address information for generating a symbol address table", wherein the generating of a symbol address table comprises finding the entry point, i.e., initialization function, for that code segment).

The motivation for combining Van Doorn and Kempf is analogous to the motivation for combining Kempf with Van Doorn, as discussed above in reference to claim 4.

As per claim 31, Kempf discloses the protection domain of claim 29, further comprising:

a linking table having a number of linking table entries, each linking table entry including a link stub directed to an address outside of the memory space (col. 7 lines 23-44, "The code table comprises entries for each linked program code segment for the address space").

10. Claims 12-14 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kempf in view of Kelly et al. (USPN 6,199,152) (hereinafter Kelly) in view of Van Doorn.

As per claim 12, Kelly and Van Doorn disclose the following limitations not specifically shown by Kempf, specifically Kelly discloses the method of claim 11, further comprising the steps of:

executing an exception handling routine in response to the generation of the processing exception, the exception handling routine including:

saving a pre-exception setting of the task protection view (col. 13 lines 39-61, "The use of such an exception handler requires that the code morphing software include routines for emulating the entire exception handling process including any hardware provided by the target computer for handling the process. This requires that the code morphing software provide for saving the state of the target processor so that it may proceed correctly after the exception has been handled").

Furthermore, Van Doorn discloses altering the task protection view to include a protection view of the second domain (col. 9 line 7-23, "memory Region 0 is mapped into the executable content context 330. Part of this memory contains executable code and has the execute privilege associated with it"); and

jumping to the external location (col. 9 lines 14-25, "Upon locating the initialization function, the client process starts a new thread of execution process linked with the program code segment and transfers control to the starting address of the initialization function").

It would have been obvious to one of ordinary skill in the art to add Kelly and Van Doorn to Kempf since saving the state of a computer prior to handling an exception allows the system to protect a known safe state. Therefore, if any error occurred during the exception handling, the system could simply roll back to the saved state. This provides an added security measure for cases when methods outside of a protection view are accessed.

As per claim 13, Van Doorn discloses the method of claim 12, wherein the task protection view is saved on a task protection switch stack (col. 4 lines 36-54, "A handler consists of a protection domain identifier, the address of a call-back function, and a stack pointer", wherein the stack pointer identifies all the namespaces within the protection view of the protection domain, and is located through the use of the protection domain identifier).

As per claim 14, Kelly discloses the method of claim 12, further comprising the steps of:

retrieving the pre-exception setting of the task protection view (col. 13 line 62 - col. 14 line 5, "In these cases, the code morphing software, using the state saving and restoring mechanisms described above, causes the target state to be restored to its most recent official version");

restoring the task protection view using the pre-exception setting of the task protection view (col. 13 line 62 - col. 14 line 5, "In these cases, the code morphing software, using the state

saving and restoring mechanisms described above, causes the target state to be restored to its most recent official version").

Furthermore, Van Doorn discloses returning to an a subsequent instruction to the jump instruction in the number of instructions (this is inherent in the disclosure of Van Doorn since it is in reference to a Java Virtual Machine that executes bytecode sequentially).

Conclusion

11. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Syed J Ali whose telephone number is (703) 305-8106. The examiner can normally be reached on Mon-Fri 8-5:30, 1st Friday off.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, William A Grant can be reached on (703) 308-1108. The fax phone numbers for the organization where this application or proceeding is assigned are (703) 746-7239 for regular communications and (703) 746-7238 for After Final communications.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703) 305-3900.

Syed Ali
August 4, 2003

MAJID BANANKHAH
PRIMARY EXAMINER